Greg Chan

EECS 349 Homework 4

Problem 1

Part A
See attached code in “hopfield.py”

Part B
See attached code in “hopfield.py”

Part C
I chose my training data by choosing the first instances of the digits 0 through 4 from the data file. I chose my examples by choosing the second instances of the digits 0 through 4. I did this so that I could make sure that my code worked properly.

For most of the examples I picked, the Hopfield network seemed to converge after two or three steps. This did converge to the right digit. I passed it the digit two as displayed on the left of Figure 1. In the first step, it changed 67 pixels. In next step, it changed 2 pixels. And on the final step it didn’t change any of the pixels. Based on this one test, it seem like it would be a reasonable method for pattern recognition. Running the Hopfield net on the same test image resulted in different steps to convergence, sometimes, it would converge after two steps and others one. Regardless, it converged to a fragmented two every time. Based on further trials with digits like 1 and 3, it seemed like the Hopfield network did not work as effectively in recognizing the digits. For numbers like 3, it converged to an answer in two steps that did not resemble any of the training examples.

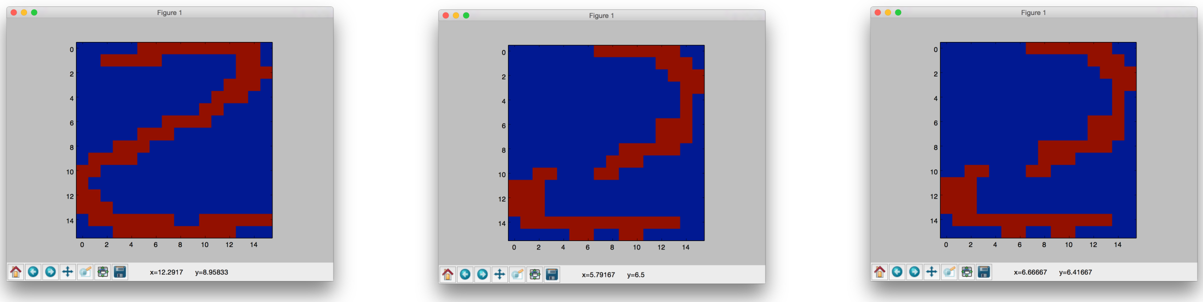


Figure 1: Progression of each step in the Hopfield network

The code I wrote for this part of the question can be seen in the function question1C(X) in the file “hopfield.py”. Part D is in question1D(X).

Part D
I trained my Hopfield net on a few pairs of digits and tested it on noisy versions of one of these training images. Every time the Hopfield net was able to converge to the correct digit. Below in Figure 2 is an illustration of an example where I trained on the digits 5 and 6.

The series looks similar but it does not take as many iterations. I was unable to find an example that would take more than two steps to converge to a solution it was trained on.

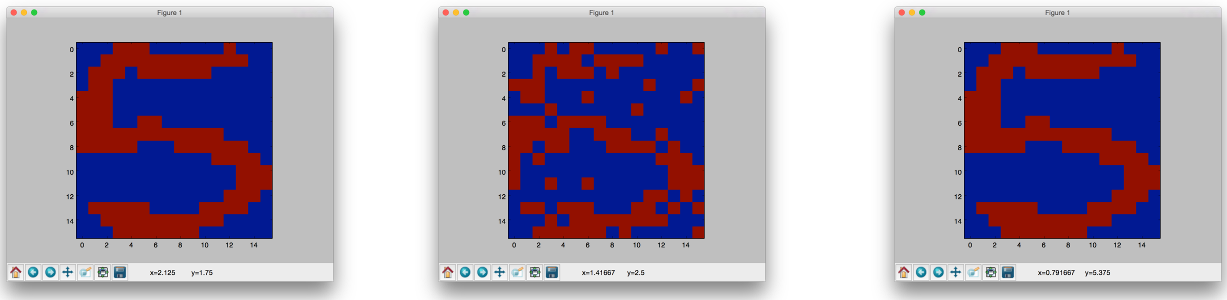


Figure 2: Progression of each step in the Hopfield network trained on 5 and 6 and tested on a noisy version of the trained example of 5

Part E

If you replace the inputs drawn from {+1, -1} with inputs drawn from {0, 1}, you get a Hopfield net that does not work as desired. Using the set of inputs drawn from {+1, -1} with the training equation as described in class (also shown in Equation 1) results in a reversed XOR of the two nodes in the network. This is because multiplying results in true (or 1) only if both are positive or both are negative. This is better described in Table 1. Using this same equation (Equation 1)

$$w_{ij} = \begin{cases} \sum_{c=1}^{C} x_{c,i} x_{c,j} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}$$

and drawing the set of inputs from {0, 1} results in a different behavior, which is more akin to AND. This is also described in Table 2.

Equation 1

A	B	A*B
-1	-1	1
-1	1	-1
1	-1	-1
1	1	1

Table 1

A	B	A*B
0	0	0
0	1	0
1	0	0
1	1	1

Table 2

To make the formulation work as desired, we would have to change the portion of the summation from $x_{c,i}x_{c,j}$ to $(2x_{c,i} - 1)(2x_{c,j} - 1)$. This would result in the function working the same as the formulation we discussed in class.

Problem 2

Part A

$$\frac{d \ln L(\theta, x)}{d\mu} = \frac{1}{\sigma^2} \sum_{i=1}^{n} (x_i - \mu)$$

$$\frac{d \ln L(\theta, x)}{d\sigma^2} = -\frac{n}{2}(\sigma^2)^{-1} + \frac{1}{2}(\sigma^2)^{-2} \sum_{i=1}^{n} (x_i - \mu)^2$$

Score vector for observation is

$$S(\theta | x_i) = \begin{pmatrix} \frac{d \ln f(\theta, x_i)}{d\mu} \\ \frac{d \ln f(\theta, x_i)}{d\sigma^2} \end{pmatrix} = \begin{pmatrix} (\sigma^2)^{-1}(x_i - \mu) \\ -\frac{1}{2}(\sigma^2)^{-1} + \frac{1}{2}(\sigma^2)^{-2}(x_i - \mu)^2 \end{pmatrix}$$

Solving for 0 gives normal equations

$$\frac{d \ln L(\theta x)}{d\mu} = \frac{1}{\hat{\sigma}^2_{MLE}} \sum_{i=1}^{n} (x_i - \hat{\mu}_{MLE}) = 0$$

$$\frac{d \ln L(\theta x)}{d\sigma^2} = -\frac{n}{2}(\sigma^2)^{-1} + \frac{1}{2}(\sigma^2)^{-2} \sum_{i=1}^{n} (x_i - \hat{\mu}_{MLE})^2 = 0$$

Solving for μ_{MLE} gives $\boxed{\hat{\mu}_{MLE} = \frac{1}{n}\sum_{i=1}^{n} x_i = \bar{x}}$

The sample average is ~~MLE~~ the MLE for μ.
Using $\hat{\mu}_{MLE} = \bar{x}$ and solving the second ~~[illegible]~~ equation yields

$$\boxed{\hat{\sigma}^2_{MLE} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2}$$

$\hat{\sigma}^2_{MLE}$ is not equal to sample variance

*Boxed answers are closed form solutions

Part B
A closed form solution is a mathematical expression that can be evaluated in a finite number of operations. Summations are allowed if the expression can be transformed from a summation to a set of elementary operations. There isn't a closed form solution to finding the mixture model that maximizes the likelihood of the data when the number of Gaussians in the Gaussian Mixture Model exceeds one because the expression used to maximize the probability of observing the data is expressed as a sum of log of sums. A sum of log of sums cannot be simplified into a closed for solution. Generally in the case of sum of logs, we can use log rules to simplify and get an analytical solution; however, because Equation 2 contains a log of sums, we cannot simplify the sum inside the log into a closed form expression. As a result, there is no closed form analytical solution to a Gaussian Mixture Model with more than one Gaussian.

Part C
For expectation maximization, we can estimate the parameters of the model and then iterate through them. The crucial part to the maximization aspect of this problem is that log-likelihood is related to the density, with respect to the same density as indexed by theta. This means that the function is maximized when the log-likelihood converges and flattens for each iteration of the algorithm. This also makes it clear that a full maximization in the maximization step is not necessary.

Part D
If the sample mean is equal to the population mean (mu equal to x), then the exponential in the Gaussian equation would be one. The update function for the Estimation Maximization would then try to minimize sigma (the standard deviation) which would force the Gaussian to shrink to one point. This is bad and to prevent this from happening, you can force a minimum sigma.

Problem 3
Part A
The homework said to put my code into two separate files; however because my code runs the training before the testing each time, I consolidated my code into one file. This file is called "gmm.py". See "gmm.py". Running this will also run all parts of the code for this question.

Part B
I don't think that my program has converged. To pick initial mu and sigma values, I visualized each class on one graph (See Figure 3), colored them differently, and estimated the values. For class 1, based on the training data, I thought I should have a Gaussian Mixture Model consisting of two Gaussians with means at 10 and 30. For class 2, based on the training data, I thought I should have a Gaussian Mixture Model consisting of three Gaussians with means at -25, -6, and 50. I chose to keep the squared standard deviations all at 1 hoping that the algorithm would correct for this. I evenly weighted all of the Gaussians. Based on the graphs of the log-likelihood of both Gaussian Mixture Models (seen in Figure 4), I think that it has converged because the line has flattened out as the number of iterations approaches 20.

Results of Training Gaussian Mixture Models
Class 1 Model
means of model 1: [9.7748859238931871, 29.582587183091857]
variance of model 1: [21.922804566237197, 9.7837696119234359]
weights of model 1: [0.59765463039946343, 0.40234536960608525]

Class 2 Model
means of model 2: [-24.82275172866651, -5.0601582832223713, 49.624444719527531]
variance of model 2: [7.9473354079336529, 23.322661814205141, 100.0243375044124]
weights of model 2: [0.20364945852814664, 0.4988430237954935, 0.29750751767685873]

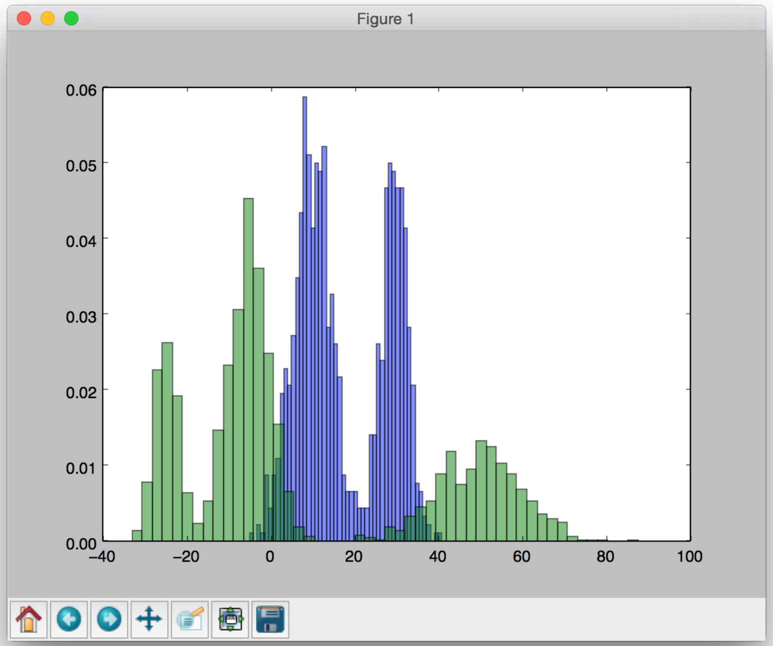
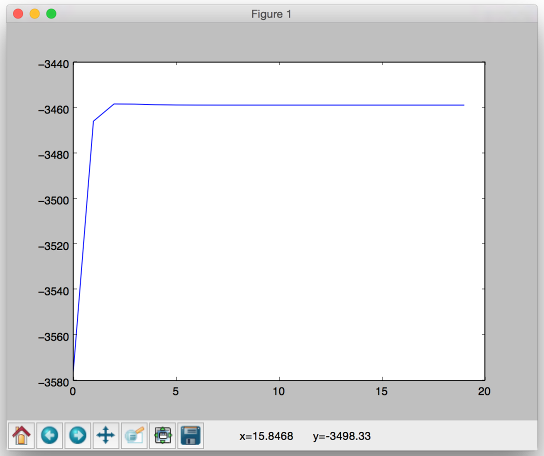


Figure 3: Class 1 in blue and Class 2 in green visualized as a histogram



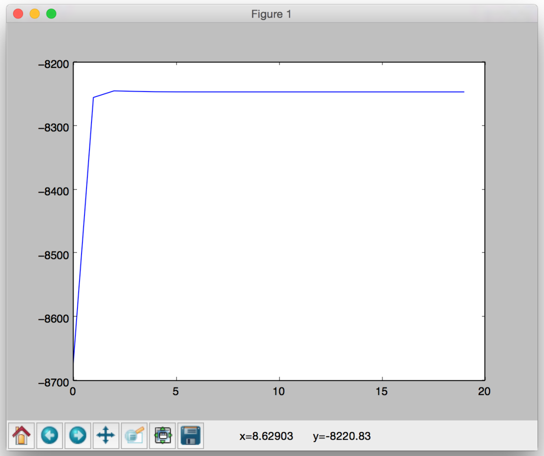


Figure 4: Left Class 1 Log-likelihood and Right Class 2 Log-likelihood

Part C
The homework said to put my code into two separate files; however because my code runs the training before the testing each time, I consolidated my code into one file. This file is called “gmm.py”. See “gmm.py”. Running this will also run all parts of the code for this question.

Part D
After running my ggmclassify on X_test with the two Gaussian Mixture Models I learned and the prior probability of p1 calculated from the training data, I received an accuracy in my classifiedr of around 93.5% when estimating the means through visualization and 92.6% by setting the means of my classifiers at the beginning all to 1. See Figure 5 for the ground-truth histogram overlaid with the classification my Gaussian Mixture Models were able to achieve.

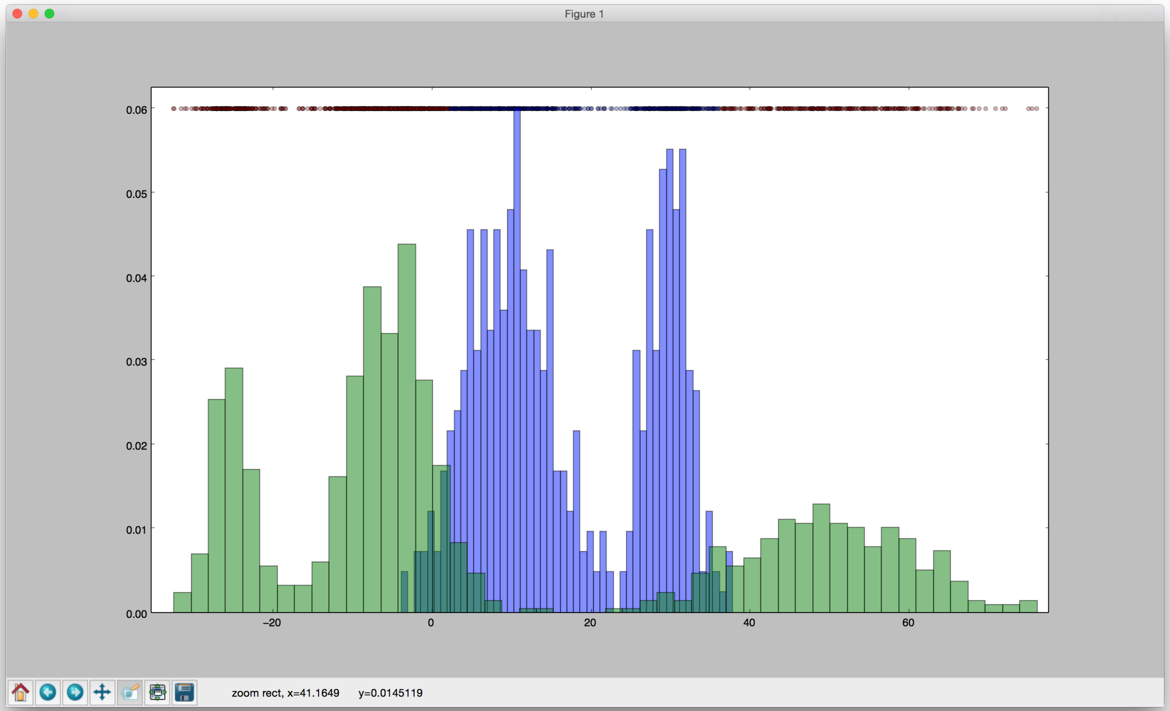


Figure 5: The Ground-truth Histogram for class 1 and class 2 overlaid with the classification of the Gaussian Mixture Model learned in part B. The red dots correspond to the green and the blue dots correspond to the blue.

Problem 4
Part A
For this question, I used the algorithm and equations described in the textbook in section 13.1. This algorithm is slightly different from the one we discussed in class because it does not include a learning rate and does not take information from the previous state with it to compute the Q function. It is also important to note that I instantiated the values of Q for each state and action pair as zero at the beginning. This is why the results of most of the movements are zero and not other random values.

Below is an on going state action pair table. I have also included a table with the final states and actions for all state action pairs used in the five adventures. Any state action pair that is left out can be assumed to have a Q value of zero. Additionally, the values from the pervious adventures are assumed to have been part of the Q table in subsequent trials (meaning values carried over from previous adventures in the Q table).

I did not use back propagation in the State Action Pair Table or in the Final State Action Pair Table; however, I have included another table that includes the result after back propagation of Q values. While this doesn't change the policy function in part B much if at all, I though it would be important to include.

State Action Pair Table

Current State	**Action**	**Q**
Adventure 1		
C1	Up	0
B1	Right	-10
Adventure 2		
C1	Up	0
B1	Up	0
A1	Right	0
A2	Right	50
Adventure 3		
C1	Right	0
C2	Up	-10
Adventure 4		
C1	Right	0
C2	Right	0
C3	Up	0
B3	Up	50
Adventure 5		
C1	Right	0
C2	Right	0
C3	Up	25
B3	Left	-10

Final State Action Pair Table

State	Action	Q
C1	Right	0
C1	Up	0
C2	Right	0
C2	Up	-10
C3	Up	25
B1	Right	-10
B1	Up	0
B3	Up	50
B3	Left	-10
A1	Right	0
A2	Right	50

State Action Pairs with Back Propagation

State	Action	Q
C1	Right	6.25
C1	Up	6.25
C2	Left	3.25
C2	Right	12.5
C2	Up	-10
C3	Up	25
B1	Right	-10
B1	Up	12.5
B3	Up	50
B3	Left	-10
A1	Right	25
A2	Right	50

Part B
A policy unction returns what action to take in state s and because the Q-learning algorithm determines the next state action pair by using an Epsilon greedy approach, depending on the value of epsilon, it will generally choose the path with the highest Q value for that state. Given the state action pairs determined in part A, we can intuitively interpret what the algorithm would do based on its greedy nature.

Policy function if epsilon is 1 without back propagated values

State	**Policy**
C1	Up or Right
C2	Left or Right
C3	Up
B1	Up or Down
B2	None
B3	Up
A1	Down or Right
A2	Right
A3	None

Policy function if epsilon is 1 with back propagated values

State	**Policy**
C1	Up or Right
C2	Right
C3	Up
B1	Up
B2	None
B3	Up
A1	Right
A2	Right
A3	None

Other formulations of epsilon would result in the policy as described above only if the chance of picking the greedy action is chosen. Otherwise, it would pick randomly.